后缀自动机
Suffix Automaton

杭州外国语学校 陈立杰
WJMZBMR

吐槽&回答

- Q：你是哪里的弱菜？我听都没听说过！
- A：我是来自杭州外国语学校的陈立杰，确实是弱菜。
- Q：Suffix Automaton？我根本就没有听说过这种数据结构。

  /li奇异夸克 2011-11-10 11:22:15

  毫不关心无压力
- A：这个还是有点用处的，等下我会讲的，你就当长知识了吧。
- Q：这有啥意思，呼噜噜～～～呼噜噜～～～
- A：睡好。。。

先让我们看SPOJ上的一道题目

- 1812. Longest Common Substring II
- 题目大意：给出N(N <= 10)个长度不超过100000的字符串，求他们的最长公共连续子串。
- 时限：SPOJ上的2s

一个简单的做法

- 二分答案之后使用哈希就可以在O(LlogL)的时间内解决这个问题。这个做法非常经典就不详细讲了。

看起来很简单。。但是。。。

Longest Common Substring II statistics & best solutions

Users accepted	Submissions	Accepted	Wrong Answer	Compile Error	Runtime Error	Time Limit Exceeded
16	750	57	129	44	89	431

All ADA ASM AWK BASH BF C C# C++ 4.3.2 C99 strict CLPS CLOJ LISP sbcl LISP clisp D ERL F# FORT GO HASK ICON
ICK JAR JAVA JS LUA NEM NICE CAML PAS fpc PAS gpc PERL PERL 6 PHP PIKE PRLG PYTH 2.5 PYTH 3.1.2 RUBY SCALA

我们可以看到大部分人都TLE了。。为什么呢？

- SPOJ太慢了
- SPOJ太慢了
- SPOJ太慢了
- SPOJ太慢了
- SPOJ太慢了
- SPOJ太慢了
- SPOJ太慢了

新的算法

2011-07-17 06:32:18 **Tony Beta Lambda**
Are we expected to implement Suffix Array or Suffix Tree?

Jin Bin: Suffix Automaton was expected.

OI中使用的的字符串处理工具

+ Suffix Array 后缀树组
+ Suffix Tree 后缀树
+ Aho-Corasick Automaton AC自动机
+ Hash 哈希

Suffix Automaton又是什么呢？

什么是自动机

- 有限状态自动机的功能是识别字符串，令一个自动机A，若它能识别字符串S，就记为A(S)=True，否则A(S)=False。
- 自动机由五个部分组成，alpha：字符集，state：状态集合，init：初始状态，end：结束状态集合，trans：状态转移函数。
- 不妨令trans(s,ch)表示当前状态是s，在读入字符ch之后，所到达的状态。

自动机识别算法

- 令将要识别的串为S
- Function recognize(String S)
- cur := init;

```
For i := 1 to Length(S)
    cur := trans(cur,S[i]);
If(cur in end) return True;
Else return False;
```

我们令Recognize(A)表示自动机A能够识别的字符串集合。也就是令函数recognize(x)为真的x的集合。

- Function recognize(State a,String S)

- cur := a;

  For i := 1 to Length(S)

  cur := trans(cur,S[i]);

  If(cur in end) return True;

  Else return False;

  这个表示从状态a开始识别串。

- 我们令Recognize(A,a)表示后缀自动机A，从状态a开始，能够识别的字符串集合。也就是令函数recognize(a,x)为真的x的集合。

后缀自动机的定义

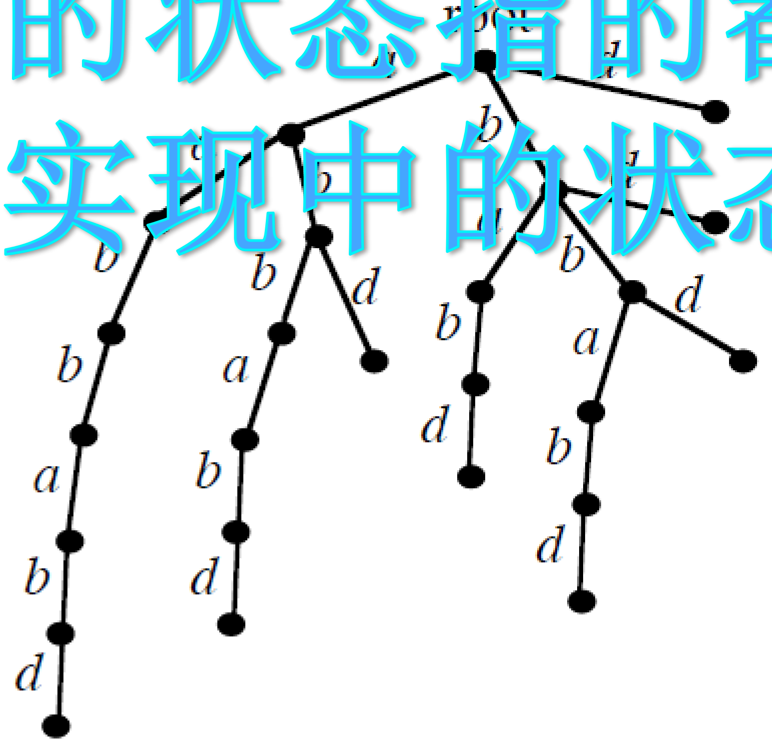
- 给定字符串S
- S的后缀自动机suffix automaton(以后简记为SAM)是一个能够识别S的所有后缀的自动机。
- 即SAM(x) = True，当且仅当x是S的后缀
- 同时后面可以看出，后缀自动机也能用来识别S所有的子串。

最简单的实现

之后我们说的状态指的都是在这个简单实现中的状态。

考虑字符串"aabbabd"

我们可以讲该字符串的所有后缀插入一个Trie中，就像右图那样。那么初始状态就是根，状态转移函数就是这颗树的边，结束状态集合就是所有的叶子。



注意到这个结构对于长度为N的串，会有O(N^2)的节点。

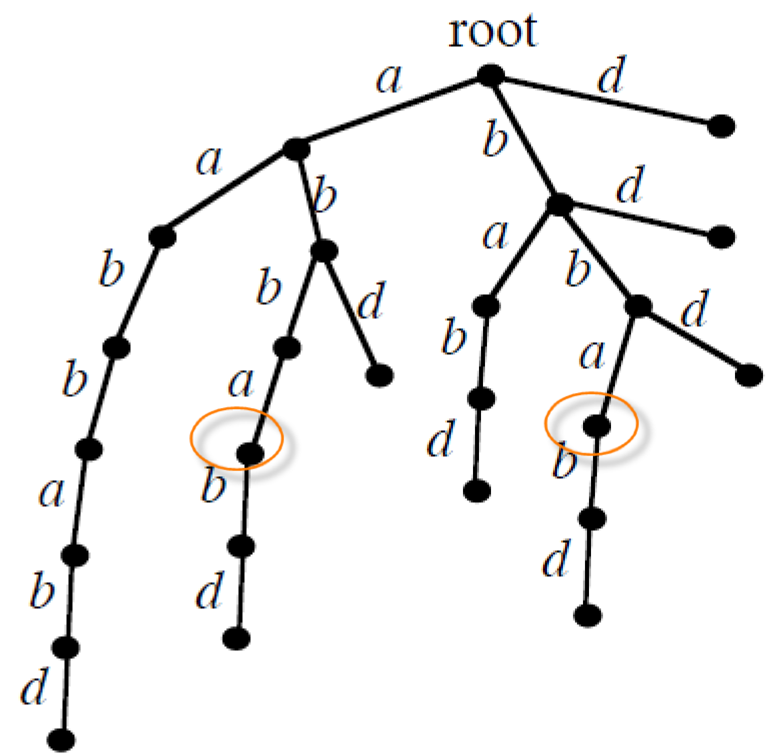
简单实现的优化

考虑字符串"aabbabd"

由于节点过多，我们为了优化时间复杂度，不妨考虑减少节点的个数。

考虑图中后缀自动机SAM的两个节点A，B。
如果Recognize(SAM,A) == Recognize(SAM,B)。也就是从A开始识别和从B开始识别是完全等价的。A和B就可以合并，以减少节点个数。

近一步我们可以发现，如果两个节点代表的子树完全相等的，它们就可以合并。比图中那两个点。



分析

- S=ABBBABBABBBBBABBA
- 简单实现中的一个节点，其到根的路径必然是S的一个子串。
- 比如状态s的前缀是BBA，那么s能够识别哪些字符串呢？
- 也就是Recognize(SAM,s)是什么？
- 对于一个后缀suffix，如果s是它的前缀，那么它就能在状态s的子树中。
- 既suffix = s + x,那么从s开始就能识别字符串x

- S=ABBBABBABBBBBABBA
- BBABBABBBBBABBA
- BBABBBBBABBA
- BBABBA
- BBA
- 考虑状态s的前缀是BBA

那么显然如果它在串中出现了N次，就有N个后缀以它为前缀。

比如一次在串中[l,r]位置出现，那么状态s开始就能识别[r+1,Length(S)]这个子串。

- 那么令s在母串S中出现了n次，分别是 $\{[l_1,r_1],[l_2,r_2],[l_3,r_3]..[l_n,r_n]\}$
- 那么 $Recognize(SAM,s)=\{[r_1+1,Length(S)],[r_2+1,Length(S)]..[r_n+1,Length(S)]\}$
- 我们可以发现，这个集合由集合 $\{r_1,r_2,r_3,...,r_n\}$ 唯一确定了，也就是说，由串s在母串中出现的结束位置集合决定了。
- 不妨令 $Right(SAM,s)=\{r_1,r_2,r_3,...,r_n\}$。

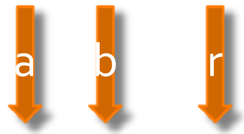
- 由于Recognize(SAM,A) == Recognize(SAM,B)等价于
- Right(SAM,A) == Right(SAM,B)
- 也就是节点A和B能够合并，当且仅当这两个A和B在字符串S中出现的结束位置集合完全一样。

优化后的表示

- 由于我们将简单表示中所有Right集合相等的节点都合并了。
- 那么现在，一个节点s就代表了母串S中的一个子串集合，它们的Right集合都是Right(s)。
- 我们不妨定义Onwer(str)为子串str所在的状态。
- 考虑一个Right集合R={r1,r2,r3...,rn}，如果子串的长度为len，那么这个子串就出现在{[r1-len+1,r1],[r2-len+1,r2]...[rn-len+1,rn]}。也就是说，一个Right集合跟一个长度，就定义了一个子串。
- 对于R，有些子串长度是合法的，有些是不合法的，不合法的情况有两种，要么是并没有出现在规定的位置，要么是不仅仅出现在了规定的位置。
- 一个显然的性质是对于R合法的子串长度，必然是一个区间。我们记该区间为[Min(s),Max(s)]

关键的性质

- 优化后的SAM表示中，最多只有O(N)个状态。我们之后SAM指的都是优化后的表示，状态也指优化后的表示中的状态。证明:由于考虑两个状态a,b,令RA=Right(a),RB=Right(b)
- 若RA和RB中有公共元素，比如r，那么由于[Min(a),Max(a)]和[Min(b),Max(b)]是不相交的，不妨令Min(a)>Max(b)，那么我们可以发现b所代表的子串全是a所代表子串的后缀。也就是说a中子串出现的结束位置，b中子串也必然出现了，那么RA就是RB的真子集。



- aaaabbbbbbbbaaaaa

关键的性质

- 那么两个状态a,b，它们的Right集合要么不相交，要么一个是另一个的真子集。

状态数既然是线性的，这个结构就有价值了。

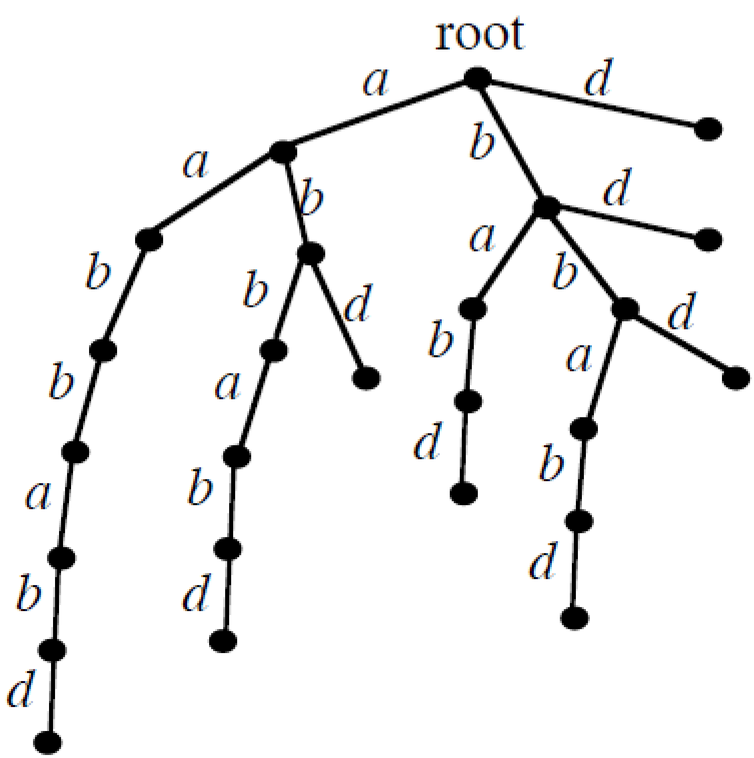
可以发现所有的状态构成了一个树型结构，由于叶子节点(就是Right集合中之有一个元素的节点)只有N个，那么总共的节点数就最多是2N个。

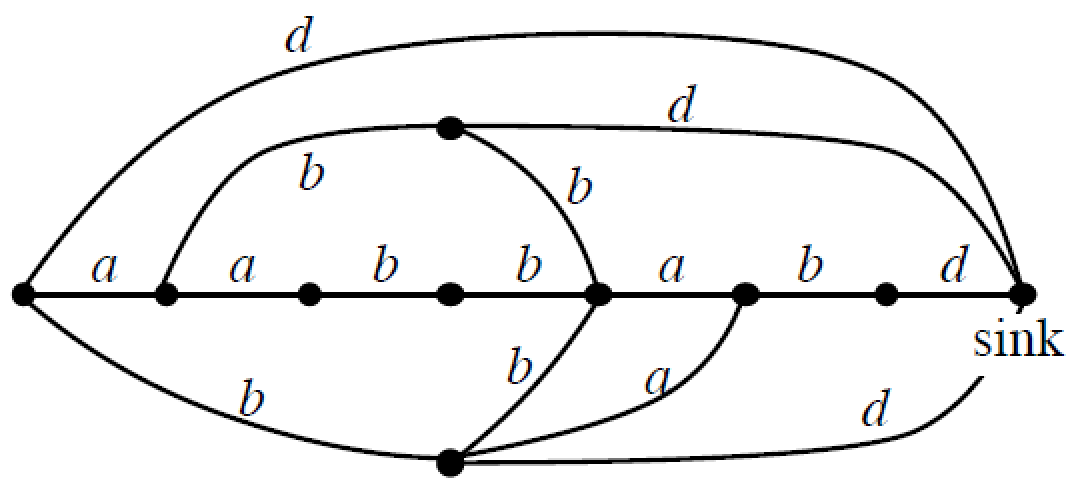
一些性质

- 令一个状态s，我们令fa=Parent(s)表示上面那个图中，它的父亲。那么fa的Right集合是s的Right集合的超集，同时fa的Right集合是这些集合中最小的。
- 考虑长度，s的范围是[Min(s),Max(s)]，考虑为什么长度Min(s)-1为什么不符合要求，可以发现肯定是因为出现的地方超出了Right(s)。同时随着长度的变小，出现的地方越来越多，那么Min(s)-1就必然属于fa的范围。那么
- Max(fa) = Min(s)-1

一些性质

- 我们已经证明了状态的个数是O(N)的，为了证明这是一个线性大小的结构，我们还需要证明trans中边的大小是O(N)的。
- 一条边a->b 有标号c，表示从a开始，读入字符c的话，就会转移到状态b。
- 我们首先求出一个自动机的生成树（注意，跟之前提到的树形结构没有关系），以init为根，这个生成树必须包含两个长度差1前缀所属的状态之间的转移。


root


sink

一些性质

- 那么令状态数为M，,生成树中的边最多只有M-1条，接下来考虑非树边。对于一条非树边(a,b,c)。
- 我们构造：
- 生成树中从根到状态a的路径+(a->b)+b到任意一个end状态。
- 可以发现这是一条从init到end状态的路径，由于这是一个识别所有后缀的后缀自动机，因此这必然是一个后缀。
- 那么一个非树边可以对应到多个后缀。我们对每个后缀，沿着自动机走，将其对应上经过的第一条非树边。
- 那么每个后缀对应的非树边都不同，同时一个非树边至少被一个后缀所对应，所以非树边的数量不会超过后缀的数量。
- 所以边的数量也不会超过O(N)

关于子串的性质

- 由于每个子串都必然包含在SAM的某个状态里。
- 那么一个字符串s是S的子串，当且仅当，从init开始，能够按顺序使用s中的字符进行转移并到达一个状态。
- 那么我们就可以用SAM来解决子串判定问题。
- 同时也可以求出这个子串的出现个数，就是所在状态Right集合的大小。

关于子串的性质

- 在一个状态中直接保存Right集合会消耗过多的空间，我们可以发现状态的Right集合就是对应的树形结构中所有的子孙中的叶子节点。
- 那么如果按dfs序排列，一个状态的right集合就是一段连续的区间中的叶子节点的编号，那么我们也就可以快速求出一个子串的所有出现位置了。

线性构造算法

- 我们的构造算法是Online的，也就是从左到右逐个添加字符串中的字符。依次构造SAM。
- 这个算法实现相比后缀树来说要简单很多，尽管可能不是非常好理解。
- 让我们先回顾一下性质

定义和性质的回顾

- 状态s，转移trans，初始状态init，结束状态集合end。
- 母串S，S的后缀自动机SAM(Suffix Automaton的缩写)。
- Right(str)表示str在母串S中所有出现的结束位置集合。
- 一个状态表示的所有子串Right集合相同,为Right(s)。
- Parent(s)表示使得Right(s)是Right(x)的真子集，并且Right(x)的大小最小的状态x。
- 一个Right集合和一个长度定义了一个子串。

定义的回顾

- 对于状态s，使得Right(s)合法的子串长度是一个区间，为[Min(s),Max(s)]
- Max(Parent(s)) = Min(s)-1。
- Parent函数可以表示一个树形结构。不妨叫它Parent树。
- SMA的状态数量和trans中边的数量，都是O(N)的。
- 不妨令Trans(s,ch)==NULL表示从s出发没有标号为ch的边，

定义的回顾

- 考虑一个状态s,它的Right集合为{r1,r2,r3..,rk}，加入有一条s->t标号为c的边，考虑t的Right集合，由于多了一个字符，s的Right集合中，只有S[r_i+1]==c的符合要求。那么t的Right集合就是{r_i+1},其中ri满足上诉条件。
- 那么如果s出发有标号为x的边，那么Parent(s)出发必然也有。
- 同时，对于令f=Parent(s)，
- Right(Trans(s,c)) ⊆ Right(Trans(f,c))。
- 有一个很显然的推论是Max(t)>Max(s)

每个阶段

- 我们每次添加一个字符，并且更新当前的SAM使得它成为包含这个新字符的SAM。
- 令当前字符串为T，新字符为x。
- SAM(T) -> SAM(Tx)
- 那么我们新增加了一些子串，它们都是串Tx的后缀。

每个阶段

- 那么我们考虑所有表示T的后缀(也就是Right集合中包含Length(T))的节点v1,v2,v3...vk。
- 由于必然存在一个Right(p)={Length(T)}的节点p(整个串T对应的节点)。那么v1,v2,v3...vk由于Right集合都含有Length(T)，那么它们在Parent树中必然全是p的祖先。可以使用Parent函数得到他们。
- 同时我们添加一个字符x后，令np表示Right(np)={Length(T)+1}的节点。
- 不妨让他们从后代到祖先排为v1=p,v2,v3....vk=root。

每个阶段

- 考虑其中一个v的Right集合{r1,r2,r3...,rn=Length(T)}。
- 那么在它的后面添加一个新字符x的话，形成新的状态nv的话，只有$S[r_i + 1] == x$的那些ri是符合要求的。
- 同时在之前我们知道，如果从v出发没有标号为x的边(我们先不看rn)，那v的Right集合内就没有满足这个要求的ri。
- 那么由于v1,v2,v3的Right集合逐渐扩大，如果vi出发有标号为x的边，那么vi+1出发也肯定有。
- 对于出发没有标号为x的边的v，它的Right集合内只有rn是满足要求的，所以根据之前提到的转移的规则，让它连一条到np标号为x的边。

每个阶段

- 令vp为v1,v2...,vk中第一有标号为x的边的。
- 考虑vp的Right集合{r1,r2,...,rn}，令trans(vp,x)=q
- 那么q的Right集合就是{ri+1}，S[ri + 1]==x的集合(注意到这是更新之前的情况，所以rn是不算的)。
- 设q的Right集合是{t1,t2,...,tm}。
- 注意到我们不一定能直接在q的Right集合中插入Length(T)+1。

每个阶段

- 最后一个是x，用红色画出vp的结束位置上，长度为Max(vp)的串。
- 用蓝色画出t的结束位置上，长度为Max(t)的串。
- 串：AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAx
- AAAAAAxAAAAAAAAAAAAAAAxAAAAAAAAAAAAAAAx

  AAAAAAxAAAAAAAAAAAAAAxAAAAAAAABAAAAAx
- 从这里可以看出，如果在Right(t)中强行插入Length(T)+1，会导致Max(t)变小。从而引发一系列的问题。
- 当然如果Max(t) == Max(vp)+1,就不会有这样的问题，我们只要让
- Parent(np)=t，就可以结束这个阶段了。

每个阶段

- 这个时候，我们注意到t的实际上被分成了两份
  AAAAAAxAAAAAAAAAAAAAAxAAAAAAAAAAAAAAx
- AAAAAAxAAAAAAAAAAAAAAxAAAAAAAABAAAAAx
- AAAAAAxAAAAAAAAAAAAAAxAAAAAAAABAAAAAx
- 那么我们新建一个节点nt，使Right(nt) = Right(t) $\cap$ $\{\mathrm{Length}(T)+1\}$
- 同时可以看出Max(nt) = Max(vp)+1。
- 那么由于Right(t)是Right(nt)的真子集，所以Parent(t) = nt。
- 同时Parent(np) = nt。
- 并且容易证明Parent(nt) = Parent(t) (原来的)

每个阶段

- 接下来考虑节点nt,在转移的过程中,结束位置Length(T)+1是不起作用的，所以trans(nt)就跟原来的trans(t)是一样的，拷贝即可。

每个阶段

- 接下来，如果新建了节点nt我们还得处理vp..vk,
- 回忆：v1,v2,…,vp,…vk是所有Right集合包含{Length(T)}的节点按后代到祖先排序，其中vp是第一个有标号为x的边的。x是这轮新加入的字符。
- 由于vp,…,vk都有标号为x的边，并且到达的点的Right集合，随着出发点Right集合的变大，也会变大，那么只有一段vp…ve，通过标号为x的边，原来是到结点t的。
  回忆：t=Trans(vp,x)。
- 那么由于在这里t节点已经被替换成了nt,我们只要把它们的Trans(*,x)设为nt即可。

每个阶段

- 自此我们圆满的解决了转移的问题。

每个阶段：回顾

- 令当前串为T，加入字符为x。
- 令p为Right(p)={Length(T)}的节点。
- 新建np表示Right(np)={Length(T)+1}的节点。
- 对p的所有没有标号x的边的祖先vi，trans(vi,x)=np
- 找到p的第一祖先vp，使得它有标号x的边，如果没有，那么Parent(p)=root，结束该阶段。
- 令t=trans(vp,x)，若Max(t)==Max(vp)+1，令Parent(np)=t，结束该阶段。

每个阶段：回顾

- 否则新建节点nt，Trans(nt,*)=Trans(t,*)
  Parent(nt) = Parent(t) (先前的)
  Parent(t) = nt
  Parent(np)=t
- 对所有Trans(vi,x) == t的vi，Trans(vi,x) 改成nt

C++的代码实现

```
struct State {
    State*par, *go[26];
    int val;
    State(int _val) :
            par(0), next(0),val(_val) {
        memset(go, 0, sizeof go);
    }
};
State*root,last;
```

C++的代码实现

```
void extend(int w) {
    State*p = last;
    State*np = new State(p->val + 1);
    while (p && p->go[w] == 0)
        p->go[w] = np, p = p->par;
    if (p == 0)
        np->par = root;
    else {
        State*q = p->go[w];
        if (p->val + 1 == q->val) {
            np->par = q;
        } else {
            State*nq = new State(p->val + 1);
            memcpy(nq->go, q->go, sizeof q->go);
            nq->par = q->par;
            q->par = nq;
            np->par = nq;
            while (p && p->go[w] == q)
                p->go[w] = nq, p = p->par;
        }
    }
    last = np;
}
```

虽然我讲了这么多

代码还是很好写的

让我们实战一下吧

I.最小循环串

- 给一个字符串S，每次可以将它的第一个字符移到最后面，球这样能得到的字典序最小的字符串。
- 如BBAAB，最小的就是AABBB
- 考虑字符串SS，我们就是要求SS的长度为Length(S)且字典序最小的子串，那么我们构造出SS的SAM，从init开始每次走标号最小的转移，走Length(S)步即可以得到结果。
- 为什么这样是对的就留给大家作为小思考了。

II.SPOJ NSUBSTR

- 给一个字符串S，令F(x)表示S的所有长度为x的子串中，出现次数的最大值。球F(1)..F(Length(S))
- Length(S) <= 250000
- 我们构造S的SAM，那么对于一个节点s，它的长度范围是[Min(s),Max(s)],同时他的出现次数是|Right(s)|。那么我们用
- |Right(s)|去更新F(Max(s))的值。
  同时最后从大到小依次用F(i)去更新F(i-1)即可。
- 为什么这样是对的也作为小思考。

III.BZOJ2555 SubString

- 你要维护一个串，支持在末尾添加一个字符和询问一个串在其中出现了多少次这两个操作。
- 必须在线。
- 构造串的SAM，每次在后面加入一个字符，可以注意到真正影响答案的是Right集合的大小，我们需要知道一个状态的Right集合有多大。

III.BZOJ2555 SubString

- 回顾构造算法，对Parent的更改每个阶段只有常数次，同时最后我们插入了状态np，就将所有np的祖先的Right集合大小+了1。
- 方法1：使用动态树维护Parent树。
  方法2：使用平衡树维护Parent树的dfs序列。
- 这两种方法跟今天的主题无关，不详细讲了。

IV:SPOJ SUBLEX

- 给一个长度不超过90000的串S，每次询问它的所有不同子串中，字典序第K小的，询问不超过500个。
- 我们可以构造出S的SAM，同时预处理从每个节点出发，还有多少不同的子串可以到达。
- 然后dfs一遍SAM，就可以回答询问了。
- 具体实现作为小练习留给大家。

V:SPOJ LCS

- 给两个长度小于100000的字符串A和B，求出他们的最长公共连续子串。
- 我们构造出A的后缀自动机SAM
- 对于SAM的每个状态s，令r为Right(s)中任意的一个元素，它代表的是结束位置在r的，长度在[Min(s),Max(s)]之间的所有子串。
  AAAAAAAAAAAAAAAAAAAArAAAAAAAAAAAAAA
  AAAAAAAAAAAAAAAAAAAAArAAAAAAAAAAAAAA
  AAAAAAAAAAAAAAAAAAAAArAAAAAAAAAAAAAA
- 我们不妨对状态s，求出所有B的子串中，从任意r开始往前能匹配的最大长度L，那么min(L,Max(s))就可以更新答案了。

V:SPOJ LCS

- 我们考虑用SAM读入字符串B。
- 令当前状态为s，同时最大匹配长度为len
- 我们读入字符x。如果s有标号为x的边，那么s=Trans(s,x),len = len+1
- 否则我们找到s的第一个祖先anc，它有标号为x的边，令s=Trans(anc,x),len=Max(anc)+1。
- 如果没有这样的祖先，那么令s=root,len=0。
- 在过程中更新状态的最大匹配长度

V:SPOJ LCS

- 注意到我们求的是对于任意一个Right集合中的r，最大的匹配长度。那么对于一个状态s，它的结果自然也可以作为它Parent的结果，我们可以从底到上更新一遍。
- 然后问题就解决了。

VI:SPOJ LCS2

- 给N个长度小于100000的字符串A和B，求出他们的最长公共连续子串。N<=10。
- 我们构造出其中一个A的后缀自动机SAM，用类似上题的方法求出每个其它串对每个状态的最大匹配长度，
- 考虑一个状态s，如果A之外其它串对它的匹配长度分别是a1,a2,..an-1，那么Min(a1,a2,..an-1,Max(s))就可以更新答案了。

一些其他的东西

- 其实不仅仅有Suffix Automaton还有。。
  Factor Automaton
- Suffix Oracle
- Factor Oracle
- Oracle的意思是神谕！听起来很强吧。

Factor Oracle

- 一个串的Factor Oracle是一个自动机，可以识别这个串的所有子串的集合，但也可以识别一些别的乱七八糟的东西。
- 其实Oracle也有预言的意思，所以这个是不一定准的。
- Factor Oracle的构造算法非常的简单，不过我也不知道这个在OI中有什么用，就不讲了。

Queries are welcomed !

广告

- 我会把课件和代码放在我的博客上，地址是：
- http://hi.baidu.com/wjbzbmr/